2ME3 - Assignment 1

Please read this document very carefully. Follow instructions exactly. If you have any questions please post them to MS Teams or ask during office hours.

This assignment is due Oct 17th, by 11:59pm

I have created an Assignment 1 channel in Teams. If you have questions about the assignment, please post them there. Thank you.
Unless specifically stated, assume you are not allowed to import external libraries.

Purpose

This assignment will have you implement a partially implemented Connect Four game. It should highlight the OO principles of encapsulation, inheritance, and polymorphism. If you are unfamiliar with Connect Four please read this article:

`https://www.wikihow.com/Play-Connect-4`

Overview

There are three Java files which accompany this document:

1. `ConnectFour.java`
2. `Player.java`
3. `Board.java`

You are responsible for submitting three Java files:

1. `Board.java`
2. `HumanPlayer.java`
3. `AIPlayer.java`

See below for details on what you are responsible for completing.

How To Begin

Read through `ConnectFour.java`, specifically, read and understand the `playGame()` method. This is where the high level logic and flow takes place. As you will see, this method is not complete due to `Player.java` being an abstract class. Study how these classes use/relate to each other and what has access to what. You may need to deduce a few things about the final implementation. The end goal of the game is to be have something as follows. If the below code was executed:

```
public static void main() {
    Board board = new Board();
    ConnectFour game = new ConnectFour(board);
    game.setPlayer1(new HumanPlayer('R', board, "Alice"));
    game.setPlayer2(new HumanPlayer('B', board, "Bob"));
    game.playGame();
}
```

then something similar to below would be output to the console. Of course, depending on user input things could vary.

```
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
|_|_|_|_|_|_|_|
It is Alice's turn.
Alice, please input your move: 3
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
|_|_|R|_|_|_|_|
It is Bob's turn.
Bob, please input your move: 3
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | |B| | | | |
|_|_|R|_|_|_|_|
It is Alice's turn.
Alice, please input your move: 4
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | |B| | | | |
|_|_|R|R|_|_|_|
It is Bob's turn.
Bob, please input your move: 4
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | |B|B| | | |
|_|_|R|R|_|_|_|
It is Alice's turn.
Alice, please input your move: 2
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | |B|B| | | |
|_|R|R|R|_|_|_|
It is Bob's turn.
Bob, please input your move: 5
```

```
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | |B|B| | | |
|_|R|R|R|B|_|_|
It is Alice's turn.
Alice, please input your move: 1
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | |B|B| | | |
|R|R|R|R|B|_|_|
"Congratulations Alice, you have won!"
```

Your Tasks

1. Create and implement two classes: `HumanPlayer.java` and `AIPlayer.java`. Both of these classes should extend `Player.java` and not be abstract. That is, they will need to implement the method `makeMove`. See the points below for information on the two different implementations.

   - For `HumanPlayer`, the `makeMove` method should prompt the user for input. You can assume the user will always input a number between 1 and 7 inclusive. However, if the user inputs an invalid move, i.e. that column is full, you should reprompt the user for a valid input.
   - For `AIPlayer.java`, the `makeMove` method can do whatever you want as long as the following two criteria are met:
     - If there is one or more winning moves available, the AI player will make one of them.
     - If there is no winning move available, but their opponent has one or more winning moves available for next turn, the AI player will block one of them.

     If you wish, you may import `java.util.Random` for this method.

2. Complete the implementation of the Board class to function as described in the How To Begin section. For the `printBoard()` method, do not worry too much about it being exactly as seen in this document, but it should be intuitive and human readable.

   You will need to have some sort of internal representation of the board within the class. How you choose to store the board state is up to you. However, this decision in no way should be exposed to any other classes. For example, if you choose to use a 2d array to store the board state, all the Player classes should not be exposed to array notation/methods in any way. In other words, if you decided to change your implementation from an array to an ArrayList, the only class you should need to modify is `Board.java`. To achieve this, you will need to add some methods which are not yet declared. If you wish, you may import external libraries for your internal board representation. For example, ArrayLists.

Submitting and Grading

This assignment will be submitted electronically via Avenue. Part of your assignment will be auto graded, part will be done manually. A rough breakdown is as follows:

- `HumanPlayer`: 20%
- `AIPlayer`: 30%
- `Board` methods: 30%
- `Board` encapsulation: 20%

Good luck!

Additional Notes

- You may import `java.util.Scanner` in the HumanPlayer class.
- Some of the interactions between the object are intentionally designed in a somewhat awkward way to get a point across. One of the main challenges you will need to address is how the AIPlayer will know if a move blocks a win without knowing the other player's symbol. You may assume during a game, both players will always have different symbols.
- For testing your game, you should use code similar to that found earlier in this document, i.e. create a Runner class with this code in it:

```
public static void main() {
    Board board = new Board();
    ConnectFour game = new ConnectFour(board);
    game.setPlayer1(new HumanPlayer('R', board, "Alice"));
    game.setPlayer2(new HumanPlayer('B', board, "Bob"));
    game.playGame();
}
```

Academic Dishonesty Disclaimer

All of the work you submit must be done by you, and your work must not be submitted by someone else. Plagiarism is academic fraud and is taken very seriously. The department uses software that compares programs for evidence of similar code.

Please don't copy. The TAs and I want you to succeed and are here to help. Here are a couple of general guidelines to help you avoid plagiarism:

Never look at another assignment solution, whether it is on paper or on the computer screen. Never show another student your assignment solution. This applies to all drafts of a solution and to incomplete solutions. If you find code on the web that solves part or all of an assignment, do not use or submit any part of it! A large percentage of the academic offenses in involve students who have never met, and who just happened to find the same solution online. If you find a solution, someone else will too.